Rony G. Flatscher

Introduction to Rexx and ooRexx

From Rexx to Open Object Rexx (ooRexx)

Edition: 1.01, 2013-11-03 (Version 101.20131103)

http://www.RonyRexx.net

c/o WU Wien, Augasse 2-6, A-1090 Wien

Acknowledgements

The author thanks the following persons for their feedback, proof reading and help in creating free Rexx-related art (in alphabetical order):

Gilbert *Barmwater* (U.S.A.): feedback, proof reading

Daniel A. *Flatscher* (Austria): proof reading

Howard *Fosdick* (U.S.A.): feedback, proof reading

René Vincent *Jansen* (The Netherlands): feedback, proof reading

Les *Koehler* (U.S.A.): feedback, proof reading

Gerald *Leitner* (Austria): feedback, proof reading

DI Walter *Pachl* (Austria): feedback, proof reading

Graham *Wilson* (South Africa): art including icons for BSF4ooRexx

Jon “Sahananda” *Wolfers* (United Kingdom): feedback, proof reading

Foreword

A Brief History of the `Rexx programming language`. In 1979 *Mike F. Cowlishaw* (MFC), an English gentleman working for IBM, devised a “human centric” programming language for the IBM mainframes that was easier to understand and to program than the arcane mainframe batch language named `Exec 2`. The design work was carried out at the IBM Research facilities in Hursley under the management of *Dr. Brian Marks*. It was probably the first time in the history of programming language design that IBMers interconnected worldwide via the IBM internal network were able to influence the design by studying the distributed specifications and giving feedback, like *Les Koehler* from IBM USA.

IBM later defined the `REXX` programming language to be the strategic batch/scripting language on all of its operating systems via IBM's `SAA` (System Application Architecture) standard. Another outstanding IBM employee who has probably been the only person to create Rexx interpreters multiple times for multiple operating systems is *Rick McGuire,* who led the development and maintenance of the `IBM SAA REXX` interpreters.

The IBM lab in Vienna created a Rexx compiler for its mainframe REXX (*Klaus Hansjakob*, *Walter Pachl*), which is being sold and maintained by IBM to this very day.

Mike F. Cowlishaw documented the REXX language in a book named “The Rexx Language” also known as “TRL” and he later became one of the few IBM Fellows[1] due to his continuing innovative and influential work (he is also attributed to be the person who made Java a strategic language and platform within IBM, having ported Java to the IBM OS/2 PC operating system in the 90'ies).

The high impact of the Rexx language can be witnessed by the appearance of numerous non-IBM implementations of the Rexx language, e.g. `Regina` (open source, *Anders Christensen*, *Mark Hessling*), `Rexx/imc` (open source, *Ian M. Collier*) or `BREXX` (open source, *Vassilis N. Vlachoudis*), but also proprietary and commercial Rexx interpreters like `ARexx` (part of the Amiga operating system), Novell Netware's Rexx (in the 90'ies), Workstation Unix

[1] An IBM Fellow is free to research and to work, very much like professors at Universities, who have the freedom to freely determine what they research and what they teach.

Rexx. Some pointers to various Rexx interpreters can be found at http://www.rexxla.org/rexxlang/mfc/rexxplat.html.

ANSI/INCITS `REXX` Standard. In 1996 the American National Standards Institute (ANSI) working group X3J18 finalized the “American National Standard for Information Systems – Programming Language REXX”. After ANSI got renamed to “INCITS (InterNational Committee for Information Technology Standards, http://www.incits.org/)” the respective standard was named “INCITS 274 :1996 [R2001]”. In 2007 the INCITS 274 REXX standard was extended for another period of ten years, reflecting the importance of the Rexx language in the industry.

The `INCITS 274 REXX` standards on decimal arithmetic served as the basis for defining decimal formats in `IEE 754-2008` and `ISO/IEC/IEEE 60559:2011`. An ANSI C implementation has been created by *Mike F. Cowlishaw* (http://www.speleotrove.com/mfc/), who on behalf of IBM has been a driving force behind standardizing decimal arithmetics in the context of IEEE, Java `JSR-13` and who also implemented an open source `decNumber` package in ANSI C.

A Brief History of the ooRexx Language. At the end of the 80'ies Dr. Brian Marks († 2012) oversaw another interesting project, named “Oryx” with the technical lead of *Simon Nash*. The aim of this project was to experiment with a Rexx interpreter that extends the Rexx language with object-oriented features. This work would later lead, under the auspices of *Rick McGuire,* to the IBM product “Object REXX”. It was first distributed with `OS/2 Warp 4` in 1997, versions for IBM AIX and MS Windows were created and sold as well by IBM.

In 2004 IBM handed the source code of “Object REXX” over to the non-profit special interest group (SIG) “Rexx Language Association (RexxLA, http://www.RexxLA.org)”. RexxLA published the first open source version of IBM's “Object REXX” as “Open Object Rexx 3.0 (http://www.ooRexx.org)” in 2005. The lead architect of this now open-source project has been *Rick McGuire*, who has been working in his own time on the open source version of ooRexx ever since.

What Is ooRexx?

- *A “classic Rexx” interpreter.* ooRexx runs any “classic Rexx” program and can be used to write “classic Rexx” programs. There is no need to use any of its new features that extend the Rexx language.

- *An object-oriented Rexx language, hence "ooRexx":* ooRexx comes with many useful classes (data types) and offers state-of-the art object-oriented features, devised in a "human-centric" way. Among other great things, ooRexx makes it easy to create multithreaded Rexx programs!
- *Fast and powerful:* ooRexx is a fast Rexx interpreter. It is a very powerful interpreter, which can be invoked from C++ or Java (via BSF4ooRexx) to allow Rexx and ooRexx macros/programs to run with C++ and Java applications. For C++ and Java applications it is possible to run multiple ooRexx interpreter instances in the same process space, each of which may execute even multithreaded Rexx code!
- *Great documentation:* IBM not only donated the source code for open sourcing to RexxLA, but also the excellent and professional technical documentation, which has been kept up-to-date. All the ooRexx documentation is available in the form of HTML and PDF-files, which can be nicely printed as books. The documentation is also directly available via the Internet: http://www.oorexx.org/docs/.
- *Free and open source:* originally created by IBM and marketed as "Object REXX", RexxLA received the sources for publishing, maintaining, and enhancing this powerful Rexx interpreter. RexxLA distributes ooRexx with source code for free: http://www.ooRexx.org.
- *Multiplatform:* ooRexx is available in 32 and 64 bit versions for the operating systems AIX, Linux, MacOSX, Windows, and can be built for any Unix implementation. Rexx programs written in one operating system environment can execute in any other operating system environment.
- *Extensible:* ooRexx comes with a powerful and easy to use C++ API which is documented in one of the accompanying ooRexx documentation PDFs (cf. rexxpg.pdf). This allows you to extend ooRexx with functions and methods implemented in C++, but also to bridge Rexx with other programming infrastructures like Java (cf. "BSF4ooRexx"). In addition it allows C++ applications to create Rexx interpreter instances which execute Rexx programs. This way it is fairly easy/simple to employ ooRexx as a macro language for any C++

applications. The BSF4ooRexx extension package provides the same functionality for Java applications.

The author wishes to acknowledge the following persons important to the Rexx world in the context of RexxLA in alphabetic order: *Gil Barmwater* (vice president), *Mike F. Cowlishaw* (honorary board member), *Chip Davis* (past president), *Mark Hessling* (board member), *René Vincent Jansen* (current president), *Les Koehler* (secretary/treasurer), *Lee Peedin* (past president), *Pam Taylor* (board member), *Jon "Sahananda" Wolfers* (board member).

Of course all the developers of ooRexx (including past) are acknowledged hereby (in alphabetic order): *David Ashley*, *Jean-Louis Faucher*, *Mark Hessling*, *Moritz Hoffmann*, *Rick McGuire*, *Mark Miesfeld*, *Lee Peedin*, *David Ruggles*, *Bruce Skelly*, *Rainer Tammer*, *Jon Wolfers*.

About this Book

This book introduces the programming language `Open Object Rexx`, also known as "`ooRexx`" in two steps:

1. Chapter 1 'The Rexx Language (“Everything Is a String”)' introduces the `Rexx` programming language that was created in 1979 by the IBM employee Mike F. Cowlishaw who later became an IBM Fellow due to his work on `Rexx`. The most important design philosophy for the language was the principle of "human-orientation", making it easy for programmers to create programs in the `Rexx` language compared to the arcane IBM mainframe batch language `Exec 2` which `Rexx` successfully replaced. One key success factor of the `Rexx` language has been its easy English-like syntax that makes it easy to learn, fast to comprehend, easy to apply and inexpensive to maintain. `Rexx` programs can be read almost like prose. As `ooRexx` is backwardly compatible to `Rexx` it can be used to learn `Rexx` and thereby the fundamentals of programming. The concepts in this chapter apply generally to all existing Rexx interpreters, which sometimes are called “classic Rexx” interpreters (as opposed to `ooRexx`, which is a leading edge `Rexx` interpreter that extends classic `Rexx` nicely into the object-oriented world). `ooRexx`-only features are highlighted in the text.

2. Chapter 2 'Extensions to the Rexx Language by ooRexx' documents the `ooRexx`-only keyword instructions (`LOOP`, `RAISE`, `USE`) and enhancements to the `Rexx` language like short hand assignment operators (e.g. “`+=`”) and the directives `::routine` and `::requires` that may prove quite helpful to “classic Rexx” programmers.

3. Chapter 3 'The ooRexx Language (“Everything Is an Object”)' builds upon the previous chapters and introduces the fundamental concepts of what is known as the "object-oriented (OO) paradigm". This is followed by an overview of the numerous new classes (data types) that come with `ooRexx` and which could be exploited by `Rexx` programmers to ease their programming life considerably in most cases. At the end of this chapter the reader should understand the OO-concepts and be able to take advantage of these new, powerful features!

4. Chapter '4 Reaching Out with ooRexx' opens with useful information about the ooRexx runtime system, followed by a categorized overview

of the ooRexx classes (data types, types) that are installed with the interpreter. The ooRexx programmer can directly use these ooRexx classes and take advantage of the features they implement.

5. Chapter 5 'Advanced Topics' introduces the interested reader to defining and implementing Rexx classes (data types), which is very easy and straight-forward. For those programmers who need the ability to create Rexx programs in which parts are executed concurrently, there is a concluding section which explains and demonstrates how easy it is to do that with ooRexx.

The structure and contents of the book are aimed at people who are interested in learning programming in Rexx and afterwards ooRexx. Still, it aims to introduce and demonstrate the concepts in a very concise, yet understandable manner. The reader is advised to consult the excellent ooRexx reference documentation, which completely documents ooRexx and is available as a nicely formatted PDF-book, named rexxref.pdf (http://www.oorexx.org/docs/rexxref/rexxref.pdf).

The way this book is written should also allow professional programmers to skim the book and learn about the fundamentals of Rexx and ooRexx by looking out for the definition boxes that are formatted like this:

> This is how a definition box is formatted. Definition boxes allow you to quickly get (re-)acquainted with the fundamental concepts that are taught in a chapter. This book will sometimes directly use the definitions of the ANSI/INCITS Rexx standard if possible in this book's context.

In addition, numerous little "nutshell programs" or "code snippets" demonstrate how to apply the introduced concepts. These programs, as short as they may seem, are full programs that can be executed as is by the ooRexx interpreter, yielding the output that is sometimes depicted alongside the program as well. "Nutshell programs" are formatted like this:

```
say "Hello world, this is Rexx speaking"
```

The above program will output the string *Hello world, this is Rexx speaking*.

Alternatively, ooRexx for Windows comes with a GUI program (menu entry named “Try Rexx (GUI)”) which allows you to enter Rexx code and execute it with the push of a button. ooRexx users on Linux or MacOSX might want to

install BSF4ooRexx[2] (`https://sourceforge.net/projects/bsf4oorexx/files/GA/`) which comes with a comparable GUI program (menu entry named "`GUI RexxTry Program (ooRexxTry.rxj)`").

Finally, ooRexx can be downloaded for free from one of the following locations:

- `http://www.oorexx.org/download.html`
- `https://sourceforge.net/projects/oorexx/files/`

There are editors that support Rexx syntax highlighting, for example the following two free and open source editors:

- "The Hessling Editor (THE)", which uses Rexx as its macro language, URL: `http://hessling-editor.sourceforge.net/`
- "vim (vi improved)", a part of many Linux distributions, is generally available for all operating systems, URL: `http://www.vim.org/`

[2] This GUI program is also available for Windows, if "BSF4ooRexx" gets installed there. "BSF4ooRexx" is an ooRexx external function package that allows ooRexx programs to interact directly with Java, which gets camouflaged as ooRexx. At the time of writing this external function package is available for Linux, MacOSX and Windows. Cf. 4.1 Exploiting Java on All Platforms, p. 157 below.

Table of Contents

List of Figures

List of Tables

List of Codes (Nutshell Examples)

List of Codes (Nutshell Examples)

List of Codes (Nutshell Examples)

Part I

The REXX Language (“Everything Is a String”)

1 The Rexx Language (“Everything Is a String”)

This chapter aims to introduce the Rexx language so as to enable the reader to use any (classic) Rexx interpreter like Regina, IBM host (mainframe) Rexx implementations and of course the ooRexx interpreter, which is compatible to Rexx. In order to achieve this goal any usage of specific ooRexx extensions (features) will be noted explicitly.

1.1 “Hello world!” in Rexx

It has become a custom to demonstrate the characteristics of a programming language by writing a small program that outputs the string *Hello world!*:

```
say "Hello world!"
```

Code 1.1-1: “Hello world!"

This small program demonstrates the English keyword instruction SAY (which indicates its purpose) and is followed by a literal string enclosed in double-quotes. Please note that Rexx is “caseless”, so it does not matter in which case the SAY keyword instruction was spelled, hence all the following spellings are fine and are taken to mean the SAY keyword instruction: SAY, Say, saY, SaY, sAY, SAy, say.

Rexx programs are made of plain text and get saved in plain (ASCII) text files with a file extension of .rex. If the above Rexx program was saved in a file named hello.rex, then this Rexx program can be executed by entering the command rexx hello.rex on the command line. After pressing the return key ↵ the Rexx interpreter loads the Rexx program from the text file hello.rex and executes it line by line. The SAY keyword instruction will output the given literal string, which will show the following result in the command line window:

```
Hello world!
```

It is also possible to run Rexx programs via the graphical user interface of the operating system by double-clicking the file directly in the folder. Alternatively, one may hover the mouse over the Rexx file, press the right mouse-button to open the context menu and choose Run from the menu. In this case the operating system will open a command line window and have

the Rexx interpreter execute the Rexx program. Upon termination of the Rexx program, the command line window will be closed automatically as well by the operating system. On modern computers this will be so fast that the user might not notice these steps, apart from maybe a brief flicker.

☞ Hint: to keep the command line window open in this use case, add the statement `PARSE PULL xyz` as the last instruction in your Rexx program. This instruction will wait for user input via the keyboard and assign the input to the indicated variable once the user presses the return key ↵. Only thereafter will the Rexx program terminate and the operating system would close the command line window. Our program may therefore be rewritten like this:

```
SAY "Hello world!"
PARSE PULL xyz
```

Running the above Rexx program will output the string *Hello world!* and then process the `PARSE PULL` keyword instruction which waits for the user to enter something via the keyboard. After pressing the return key ↵ the `PARSE PULL` instruction will assign the keyboard input to the variable `xyz` and as there are no more instructions left in the program the Rexx interpreter will stop executing and return control to the operating system.

1.2 Fundamental Language Concepts

The author of the Rexx language, Michael F. Cowlishaw, discusses the design principles of the REXX language in his book “The REXX Language”:[3]

- *Readability:* the structure of the syntax and the names of the keyword instructions should be easily readable. Although Rexx does not distinguish between lower- and uppercase, one can use case to make programs more legible.
- *Natural data typing:* unlike many languages, Rexx is not strongly typed. Everything is a string and its meaning depends only on its usage. For example, arithmetic operations would cause the interpreter to check whether the string operands contained valid numbers to carry out the desired arithmetic and would raise a runtime error if not.

[3] The following list of language concepts is taken from `http://speleotrove.com/rexxhist/TRL-background.pdf` (as of March 2013), made available to the public by Mike F. Cowlishaw .

- *Emphasis on symbolic manipulation:* as everything in Rexx is a string there is a rich set of string manipulation operators and functions. One of the most common string operations is concatenation for which Rexx supplies two different means: concatenating strings by intervening white space (space or tabulator character) which becomes part of the concatenated string or by an explicit concatenation operator (two vertical bars, `||`) that concatenates two strings without an intervening blank.
- *Dynamic scoping:* Rexx scoping adheres to the execution sequence of Rexx clauses as defined by the programmer.
- *Nothing to declare:* there is no mechanism for defining variables. If a variable is needed one can use a variable symbol.
- *System independence:* the language is defined in a manner which makes it independent of the hardware and system software (although it is possible to interact with the environment).
- *Limited span syntactic units:* syntactic units in Rexx are usually `clauses`, which span a single line only. If an error is detected or a program is traced, then these units are used to give the programmer good diagnostics.
- *Dealing with reality:* although consistency has been a major design goal, if in real use it creates unexpected side effects, then these were addressed in the language's design as was the case with the `TRACE` keyword instruction.
- *Be adaptable:* Rexx does not reserve any keywords, such that the language can be adapted without unexpected side effects should new keyword instructions be needed.
- *Keep the language small:* features were only added, if they were of use for a significant number of users. A small language can be learned fast and memorized for a long time.
- *No defined size or shape limits:* the Rexx language does not define any limits on the size or shape of its *TOKENS*[4] or data.[5]

[4] A *TOKEN* ist the unit of low-level syntax from which one constructs Rexx clauses. Cf. subsection ??, p. 45 below.

[5] Different implementations of Rexx interpreters may have implementation dependent limits.

All of these concepts should foster the definition of a “human centric” language.

1.3 Building Blocks and Definitions

This section defines the allowable characters to create Rexx programs and the building blocks of a Rexx program.

1.3.1 Characters Allowed in a Rexx Program

A Rexx program may consist of the following characters:

- alphabetical lowercase characters: `abcdefghijklmnopqrstuvwxyz`
- alphabetical uppercase characters: `ABCDEFGHIJLMNOPQRSTUVWXYZ`
- digit characters: `0123456789`
- the following characters: `!` (exclamation mark), `?` (question mark), `_` (underscore)
- the following *special characters* for defining clauses and expressions (each of the following characters is regarded to be a *TOKEN* that can be distinguished from any other *TOKEN* in a clause): `;` (semicolon), `:` (colon), `.` (dot), `+` (plus), `-` (minus), `*` (asterisk), `/` (slash) `,` (comma), `=` (equal sign), `|` (vertical bar), `\` (backslash), `¬` (“not” character), `(` (open parenthesis), `)` (close parenthesis)
- in addition, the following *special characters* in `ooRexx` (each of the following characters is regarded to be a *TOKEN* that can be distinguished from any other *TOKEN* in a clause): `~` (tilde), `[` (square open parenthesis), `]` (square close parenthesis)
- whitespace (non-visible) characters: space/blank character (`␣`) and tabulator character (`➔`)

In addition literal (quoted) strings and comments may contain any character.

The allowable characters can be used for different purposes when writing Rexx programs.